ESORM - ElasticSearch ORM

Release 0.3.2

Adam Wallner

May 04, 2024

CONTENTS

sphinx-quickstart on Fri Oct 20 18:53:46 2023. You can adapt this file completely to your liking, but it should at least contain the root *toctree* directive.

CHAPTER

ONE

ESORM - PYTHON ELASTICSEARCH ORM BASED ON PYDANTIC

Some ideas come from Pydastic library, which is similar, but not as advanced (yet).

1.1 Table of Contents

1.2 Installation

```
pip install pyesorm
```

1.3 Features

- Pydantic model representation of ElasticSearch documents
- Automatic mapping and index creation
- CRUD operations
- Full async support (no sync version at all)
- Mapping to and from ElasticSearch types
- Support for nested documents
- Custom id field
- Context for bulk operations
- Supported IDE autocompletion and type checking (PyCharm tested)
- Everything in the source code is documented and annotated
- `TypedDict`s for ElasticSearch queries and aggregations
- Docstring support for fields
- Shard routing support
- Lazy properties
- Support >= Python 3.8 (tested with 3.8 through 3.12)

- Support for ElasticSearch 8.x and 7.x
- Watcher support (You may need ElasticSearch subscription license for this)
- Pagination and sorting
- FastAPI integration

Not all ElasticSearch features are supported yet, pull requests are welcome.

1.3.1 Supported ElasticSearch versions

It is tested with ElasticSearch 7.x and 8.x.

1.3.2 Supported Python versions

Tested with Python 3.8 through 3.12.

1.4 Usage

1.4.1 Define a model

You can use all Pydantic model features, because `ESModel` is a subclass of `pydantic.BaseModel`. (Actually it is a subclass of `ESBaseModel`, see more *below...*)

`ESModel` extends pydantic `BaseModel` with ElasticSearch specific features. It serializes and deserializes documents to and from ElasticSearch types and handle ElasticSearch operations in the background.

Python basic types

```
from esorm import ESModel


class User(ESModel):
    name: str
    age: int
```

This is how the python types are converted to ES types:

Python type	ES type
str	text
int	long
float	double
bool	boolean
datetime.datetime	date
datetime.date	date
datetime.time	date

ESORM field types

You can specify ElasticSearch special fields using `esorm.fields` module.

```
from esorm import ESModel
from esorm.fields import keyword, text, byte, geo_point


class User(ESModel):
    name: text
    email: keyword
    age: byte
    location: geo_point
    ...
```

The supported fields are:

Field name	ES type
keyword	keyword
text	text
binary	binary
byte	byte
short	short
integer or int32	integer
long or int64	long
float16 or half_float	half_float
float32	float
double	double
boolean	boolean
geo_point	geo_point

The `binary` field accepts **base64** encoded strings. However, if you provide `bytes` to it, they will be automatically converted to a **base64** string during serialization. When you retrieve the field, it will always be a **base64** encoded string. You can easily convert it back to bytes using the `bytes()` method: `binary_field.bytes()`.

Nested documents

```
from esorm import ESModel
from esorm.fields import keyword, text, byte


class User(ESModel):
    name: text
    email: keyword
    age: byte = 18


class Post(ESModel):
    title: text
    content: text
    writer: User  # User is a nested document
```

ESBaseModel

`ESBaseModel` is the base of `ESModel`, also it is useful to use it for nested documents, because by using it will not be included in the ElasticSearch index.

```
from esorm import ESModel, ESBaseModel
from esorm.fields import keyword, text, byte


# This way `User` model won't be in the index
class User(ESBaseModel):  # <---------------
    name: text
    email: keyword
    age: byte = 18


class Post(ESModel):
    title: text
    content: text
    writer: User  # User is a nested document
```

Id field

You can specify id field in *model settings*:

```
from esorm import ESModel
from esorm.fields import keyword, text, byte


class User(ESModel):
    class ESConfig:
        id_field = 'email'

    name: text
    email: keyword
    age: byte = 18
```

This way the field specified in `id_field` will be removed from the document and used as the document `_id` in the index.

If you specify a field named `id` in your model, it will be used as the document `_id` in the index (it will automatically override the `id_field` setting):

```
from esorm import ESModel


class User(ESModel):
    id: int  # This will be used as the document _id in the index
    name: str
```

You can also create an `__id__` property in your model to return a custom id:

```
from esorm import ESModel
from esorm.fields import keyword, text, byte


class User(ESModel):
    name: text
    email: keyword
    age: byte = 18

    @property
    def __id__(self) -> str:
        return self.email
```

NOTE: annotation of `__id__` method is important, and it must be declared as a property.

Model Settings

You can specify model settings using `ESConfig` child class.

```
from typing import Optional, List, Dict, Any
from esorm import ESModel


class User(ESModel):
    class ESConfig:
        """ ESModel Config """
        # The index name
        index: Optional[str] = None
        # The name of the 'id' field
        id_field: Optional[str] = None
        # Default sort
        default_sort: Optional[List[Dict[str, Dict[str, str]]]] = None
        # ElasticSearch index settings (https://www.elastic.co/guide/en/elasticsearch/
↪reference/current/index-modules.html)
        settings: Optional[Dict[str, Any]] = None
        # Maximum recursion depth of lazy properties
        lazy_property_max_recursion_depth: int = 1
```

ESModelTimestamp

You can use `ESModelTimestamp` class to add `created_at` and `updated_at` fields to your model:

```
from esorm import ESModelTimestamp


class User(ESModelTimestamp):
    name: str
    age: int
```

These fields will be automatically updated to the actual `datetime` when you create or update a document. The `created_at` field will be set only when you create a document. The `updated_at` field will be set when you create or update a document.

Describe fields

You can use the usual `Pydantic` field description, but you can also use docstrings like this:

```
from esorm import ESModel
from esorm.fields import TextField


class User(ESModel):
    name: str = 'John Doe'
    """ The name of the user """
```

(continues on next page)

(continued from previous page)

```
    age: int = 18
    """ The age of the user """

    # This is the usual Pydantic way, but I think docstrings are more intuitive and␣
↪readable
    address: str = TextField(description="The address of the user")
```

The documentation is usseful if you create an API and you want to generate documentation from the model. It can be used in FastAPI for example.

1.4.2 Connecting to ElasticSearch

You can connect with a simple connection string:

```
from esorm import connect


async def es_init():
    await connect('localhost:9200')
```

Also you can connect to multiple hosts if you have a cluster:

```
from esorm import connect


async def es_init():
    await connect(['localhost:9200', 'localhost:9201'])
```

You can wait for node or cluster to be ready (recommended):

```
from esorm import connect


async def es_init():
    await connect('localhost:9200', wait=True)
```

This will ping the node in 2 seconds intervals until it is ready. It can be a long time.

You can pass any arguments that `AsyncElasticsearch` supports:

```
from esorm import connect


async def es_init():
    await connect('localhost:9200', wait=True, sniff_on_start=True, sniff_on_connection_
↪fail=True)
```

Client

The `connect` function is a wrapper for the `AsyncElasticsearch` constructor. It creates and stores a global instance of a proxy to an `AsyncElasticsearch` instance. The model operations will use this instance to communicate with ElasticSearch. You can retrieve the proxy client instance and you can use the same way as `AsyncElasticsearch` instance:

```
from esorm import es


async def es_init():
    await es.ping()
```

1.4.3 Create index templates

You can create index templates easily:

```
from esorm import model as esorm_model


# Create index template
async def prepare_es():
    await esorm_model.create_index_template('default_template',
                                            prefix_name='esorm_',
                                            shards=3,
                                            auto_expand_replicas='1-5')
```

Here this will be applied all `esorm_` prefixed (default) indices.

All indices created by ESORM have a prefix, which you can modify globally if you want:

```
from esorm.model import set_default_index_prefix

set_default_index_prefix('custom_prefix_')
```

The default prefix is `esorm_`.

1.4.4 Create indices and mappings

You can create indices and mappings automatically from your models:

```
from esorm import setup_mappings


# Create indices and mappings
async def prepare_es():
    import models  # Import your models
    # Here models argument is not needed, but you can pass it to prevent unused import␣
↪warning
    await setup_mappings(models)
```

First you must create (import) all model classes. Model classes will be registered into a global registry. Then you can call `setup_mappings` function to create indices and mappings for all registered models.

IMPORTANT: This method will ignore mapping errors if you already have an index with the same name. It can update the indices by new fields, but cannot modify or delete fields! For that you need to reindex your ES database. It is an ElasticSearch limitation.

1.4.5 CRUD: Create

```python
from esorm import ESModel


# Here the model have automatically generated id
class User(ESModel):
    name: str
    age: int


async def create_user():
    # Create a new user
    user = User(name='John Doe', age=25)
    # Save the user to ElasticSearch
    new_user_id = await user.save()
    print(new_user_id)
```

1.4.6 CRUD: Read

```python
from esorm import ESModel


# Here the model have automatically generated id
class User(ESModel):
    name: str
    age: int


async def get_user(user_id: str):
    user = await User.get(user_id)
    print(user.name)
```

1.4.7 CRUD: Update

```
from esorm import ESModel


# Here the model have automatically generated id
class User(ESModel):
    name: str
    age: int


async def update_user(user_id: str):
    user = await User.get(user_id)
    user.name = 'Jane Doe'
    await user.save()
```

1.4.8 CRUD: Delete

```
from esorm import ESModel


# Here the model have automatically generated id
class User(ESModel):
    name: str
    age: int


async def delete_user(user_id: str):
    user = await User.get(user_id)
    await user.delete()
```

1.4.9 Bulk operations

Bulk operations could be much faster than single operations, if you have lot of documents to create, update or delete.

You can use context for bulk operations:

```
from typing import List
from esorm import ESModel, ESBulk


# Here the model have automatically generated id
class User(ESModel):
    name: str
    age: int


async def bulk_create_users():
```

(continues on next page)

(continued from previous page)

```
    async with ESBulk() as bulk:
        # Creating or modifiying models
        for i in range(10):
            user = User(name=f'User {i}', age=i)
            await bulk.save(user)


async def bulk_delete_users(users: List[User]):
    async with ESBulk(wait_for=True) as bulk:  # Here we wait for the bulk operation to␣
↪finish
        # Deleting models
        for user in users:
            await bulk.delete(user)
```

The `wait_for` argument is optional. If it is `True`, the context will wait for the bulk operation to finish.

1.4.10 Search

General search

You can search for documents using `search` method, where an ES query can be specified as a dictionary. You can use `res_dict=True` argument to get the result as a dictionary instead of a list. The key will be the `id` of the document: `await User.search(query, res_dict=True)`.

If you only need one result, you can use `search_one` method.

```
from esorm import ESModel


# Here the model have automatically generated id
class User(ESModel):
    name: str
    age: int


async def search_users():
    # Search for users at least 18 years old
    users = await User.search(
        query={
            'bool': {
                'must': [{
                    'range': {
                        'age': {
                            'gte': 18
                        }
                    }
                }]
            }
```

(continues on next page)

(continued from previous page)

```
        }
    )
    for user in users:
        print(user.name)


async def search_one_user():
    # Search a user named John Doe
    user = await User.search_one(
        query={
            'bool': {
                'must': [{
                    'match': {
                        'name': {
                            'query': 'John Doe'
                        }
                    }
                }]
            }
        }
    )
    print(user.name)
```

Queries are type checked, because they are annotated as `TypedDicts`. You can use IDE autocompletion and type checking.

Search with field value terms (dictionary search)

You can search for documents using `search_by_fields` method, where you can specify a field and a value. It also has a `res_dict` argument and `search_one_by_fields` variant.

```
from esorm import ESModel


# Here the model have automatically generated id
class User(ESModel):
    name: str
    age: int


async def search_users():
    # Search users age is 18
    users = await User.search_by_fields({'age': 18})
    for user in users:
        print(user.name)
```

1.4.11 Aggregations

You can use `aggregate` method to get aggregations. You can specify an ES aggregation query as a dictionary. It also accepts normal ES queries, to be able to fiter which documents you want to aggregate. Both the aggs parameter and the query parameter are type checked, because they are annotated as `TypedDict`s. You can use IDE autocompletion and type checking.

```
from esorm import ESModel

# Here the model have automatically generated id
class User(ESModel):
    name: str
    age: int
    country: str

async def aggregate_avg():
    # Get average age of users
    aggs_def = {
        'avg_age': {
            'avg': {
                'field': 'age'
            }
        }
    }
    aggs = await User.aggregate(aggs_def)
    print(aggs['avg_age']['value'])

async def aggregate_avg_by_country(country = 'Hungary'):
    # Get average age of users by country
    aggs_def = {
        'avg_age': {
            'avg': {
                'field': 'age'
            }
        }
    }
    query = {
        'bool': {
            'must': [{
                'match': {
                    'country': {
                        'query': country
                    }
                }
            }]
        }
    }
    aggs = await User.aggregate(aggs_def, query)
    print(aggs['avg_age']['value'])


async def aggregate_terms():
    # Get number of users by country
```

(continues on next page)

(continued from previous page)

```
    aggs_def = {
        'countries': {
            'terms': {
                'field': 'country'
            }
        }
    }
    aggs = await User.aggregate(aggs_def)
    for bucket in aggs['countries']['buckets']:
        print(bucket['key'], bucket['doc_count'])
```

1.4.12 Pagination and sorting

You can use `Pagination` and `Sort` classes to decorate your models. They simply wrap your models and add pagination and sorting functionality to them.

Pagination

```
from esorm.model import ESModel, Pagination


class User(ESModel):
    id: int  # This will be used as the document _id in the index
    name: str
    age: int


def get_users(page = 1, page_size = 10):
    # 1st create the decorator itself
    pagination = Pagination(page=page, page_size=page_size)

    # Then decorate your model
    res = pagination(User).search_by_fields(age=18)

    # Here the result has maximum 10 items
    return res
```

Sorting

It is similar to pagination:

```
from esorm.model import ESModel, Sort


class User(ESModel):
    id: int  # This will be used as the document _id in the index
    name: str
```

(continues on next page)

(continued from previous page)

```
    age: int


def get_users():
    # 1st create the decorator itself
    sort = Sort(sort=[
        {'age': {'order': 'desc'}},
        {'name': {'order': 'asc'}}
    ])

    # Then decorate your model
    res = sort(User).search_by_fields(age=18)

    # Here the result is sorted by age ascending
    return res

def get_user_sorted_by_name():
    # You can also use this simplified syntax
    sort = Sort(sort='name')

    # Then decorate your model
    res = sort(User).all()

    # Here the result is sorted by age descending
    return res
```

1.5 Testing

For testing you can use the `test.sh` in the root directory. It is a script to running tests on multiple python interpreters in virtual environments. At the top of the file you can specify which python interpreters you want to test. The ES versions are specified in `tests/docker-compose.yml` file.

If you already have a virtual environment, simply use `pytest` to run the tests.

1.6 License

This project is licensed under the terms of the Mozilla Public License 2.0 (MPL 2.0) license.

1.7 Citation

If you use this project in your research, please cite it using the following BibTeX entry:

```
@misc{esorm,
  author = {Adam Wallner},
  title = {ESORM: ElasticSearch Object Relational Mapper},
  year = {2023},
  publisher = {GitHub},
  journal = {GitHub repository},
  howpublished = {\url{https://github.com/wallneradam/esorm}},
}
```

CHAPTER

TWO

ADVANCED USAGE

2.1 Lazy properties

ESORM is based on pydantic which is fully synchronous, so it's not possible to use async functions to calculate property values. Because of this, you also can't query another model in a computed field. To solve this problem, ESORM provides lazy properties.

You can create lazy properties like this:

```
from typing import List
from esorm import ESModel, lazy_property
from pydantic import computed_field

class User(ESModel):
    first_name: str
    last_name: str

    # This is classic pydantic computed field, which can be only synchronous
    @computed_field
    @property
    def full_name(self) -> str:
        return f"{self.first_name} {self.last_name}"

    # This is lazy property, which can be async
    @lazy_property
    async def same_first_name(self) -> List["User"]:
        return await self.search_by_fields(first_name=self.first_name)
```

Lazy properties in the background work like the following:

- they are registered in the model by storing the async function
- replace it to a real property on model creation
- after a query is executed (which is always async), lazy properties are calculated and stored in the model
- these stored values are used when accessing the property, e.g, when they are serialized to JSON

Lazy properties are computed parallelly, you can configure the number of parallel query tasks by `set_max_lazy_property_concurrency` function:

```python
from esorm.model import set_max_lazy_property_concurrency

set_max_lazy_property_concurrency(10)  # Set the number of parallel tasks to 10
```

The above example is recursive, because `User` model is used in the `same_first_name` property. This could lead to an infinite loop. Because of this, ESORM restricts the depth of recursion to 1 by default.

If you want to change the recursion depth, you can do it by setting the `max_lazy_property_depth` in the `ESConfig`:

```python
from esorm import ESModel


class User(ESModel):
    class ESConfig:
        lazy_property_max_recursion_depth = 2  # Set the recursion depth to 2

    first_name: str
    last_name: str
    ...
```

2.2 Shard routing

Shard routing is a feature of Elasticsearch which allows you to store documents in a specific shard. This can be useful if you want to store documents of a specific type in a specific shard, e.g, you want to store all documents of a specific region. When using shard routing, ElasticSearch does not need to search all shards, but only the shards which contain the documents you are looking for.

More info: https://www.elastic.co/guide/en/elasticsearch/reference/current/search-shard-routing.html

In ESORM shard routing looks like this:

```python
from typing import List
from esorm import ESModel

class User(ESModel):
    first_name: str
    last_name: str
    region: str

    @property
    def __routing__(self) -> str:
        """ Return the routing value for this document """
        return self.region + '_routing'  # Calculate the routing value from the region␣
↪field

async def get_user_by_region(region: str = 'europe') -> List[User]:
    """ Search for users by region using shard routing """
    return await User.search_by_fields(region=region, routing=f"{region}_routing")
```

2.3 Watchers

You can add watches to automatically perform an action when certain conditions are met. The conditions are generally based on data you've loaded into the watch, also known as the Watch Payload. This payload can be loaded from different sources - from Elasticsearch, an external HTTP service, or even a combination of the two.

More info: https://www.elastic.co/guide/en/elasticsearch/reference/current/how-watcher-works.html

The following example shows how to create a watcher which deletes all draft documents older than 1 hour:

```
from esorm.watcher import DeleteWatcher
from esorm import query

TIMEOUT = 60 * 60  # 1 hour


class PurgeDraft(DeleteWatcher):
    """
    Purge draft data after TIMEOUT
    """
    trigger = {
        "schedule": {
            "interval": f"30s"
        }
    }

    _index = f"draft"
    _query: query.ESQuery = {
        "bool": {
            "must": [
                # Search for all documents with id starting with "_"
                {
                    "wildcard": {
                        "id": {
                            "value": "_*"
                        }
                    }
                },
                # Filter documents which are older than TIMEOUT + 30s
                {
                    "range": {
                        "created_at": {
                            "lt": f"now-{TIMEOUT + 30}s",  # 30 sec buffer
                        }
                    }
                }
            ]
        }
    }
```

For more info check the reference, or ElasticSearch documentation, or the source code.

2.4 FastAPI integration

Because ESORM is based on pydantic, it can be easily integrated with FastAPI:

```
from typing import List, Optional
from esorm import ESModelTimestamp
from fastapi import FastAPI


class User(ESModelTimestamp):
    """ The User model """
    first_name: str
    last_name: str


app = FastAPI()


@app.post("/users")
async def create_user(first_name: str, last_name: str) -> User:
    """ Create a new user """
    user = User(first_name=first_name, last_name=last_name)
    await user.save()
    return user


@app.get("/users")
async def users(first_name: Optional[str] = None, last_name: Optional[str] = None) ->␣
↪List[User]:
    """ Search users """
    return await User.search_by_fields(first_name=first_name, last_name=last_name)
```

2.4.1 FastAPI pagination

You can add pagination and sort parameters as a dependency in endpoint arguments:

```
from typing import List
from esorm import ESModelTimestamp, Pagination, Sort
from fastapi import FastAPI, Depends

from esorm.fastapi import make_dep_sort, make_dep_pagination


class User(ESModelTimestamp):
    """ The User model """
    first_name: str
    last_name: str


app = FastAPI()

@app.get("/all_users")
```

(continues on next page)

(continued from previous page)

```
async def all_users(
    # This will create a _page, and a _page_size query parameter for the endpoint
    pagination: Pagination = Depends(make_dep_pagination(default_page=1, default_page_
↪size=10)),
    # This will create a _sort enum query parameter for the endpoint, so it is␣
↪selectable in swagger UI
    sort: Sort = Depends(make_dep_sort(
        first_name_last_name_asc=  # This is the name of the 1st sort option
        # Definition of the sort options
        [
            {'first_name': {"order": "asc"}},
            {'last_name': {"order": "asc"}},
        ],
        last_name_first_name_asc=  # This is the name of the 2nd sort option
        # Definition of the sort options
        [
            {'last_name': {"order": "asc"}},
            {'first_name': {"order": "asc"}},
        ]
    )),
) -> List[User]:
    """ Get all users """
    return await sort(pagination(User)).all()
```

CHAPTER

THREE

REFERENCE

3.1 Module contents

ESORM is an elasticsearch python ORM based on Pydantic

class esorm.**ESBaseModel**(***data*)

Bases: `BaseModel`

Base class for Elastic

It is useful for nested models, if you don't need the model in ES mappings

class ESConfig

Bases: `object`

ESBaseModel Config

This is just for lazy properties, to make ESBasemodel compatible with them

lazy_property_max_recursion_depth: int = 1

Maximum recursion depth of lazy properties

async calc_lazy_properties()

(re)Calculate lazy properties

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'forbid', 'populate_by_name': True, 'ser_json_bytes': 'base64', 'str_strip_whitespace': True, 'validate_assignment': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class esorm.**ESBulk**(*wait_for=False*, ***bulk_kwargs*)

Bases: `object`

Bulk operation for ElasticSearch

async delete(*model*)

Add the model to the bulk for deletion :type model: TypeVar(TModel, bound= ESModel) :param model: The model to add for deletion

async save(*model*)

Add the model to the bulk for saving :type model: TypeVar(TModel, bound= ESModel) :param model: The model to add for saving

class esorm.**ESModel**(***data*)

Bases: *ESBaseModel*

ElasticSearch Base Model

class ESConfig

Bases: object

ESModel Config

default_sort: Optional[List[Dict[str, Dict[str, str]]]] = None

Default sort

id_field: Optional[str] = None

The name of the 'id' field

index: Optional[str] = None

The index name

lazy_property_max_recursion_depth: int = 1

Maximum recursion depth of lazy properties

settings: Optional[Dict[str, Any]] = None

Index settings

async classmethod aggregate(*aggs*, ***, *query=None*, *routing=None*, ***kwargs*)

Aggregate Model with aggregation dict Before aggregation the model can be filtered by query dict.

Parameters

- **aggs** (Dict[str, *ESAgg*]) – Aggregation dict
- **query** (Optional[*ESQuery*]) – ElasticSearch query dict
- **routing** (Optional[str]) – Shard routing value
- **kwargs** – Other search API params

Return type

Dict[str, Union[*ESAggValueResponse*, *ESAggTermsResponse*, *ESAggHistogramResponse*]]

Returns

The result list

async classmethod all(***kwargs*)

Get all documents

Parameters

kwargs – Other search API params

Return type

List[TypeVar(TModel, bound= ESModel)]

Returns
The result list

async classmethod call(*method_name*, *, *wait_for=None*, **kwargs*)

Call an elasticsearch method

This is a low level ES method call, it is not recommended to use this directly.

Parameters

- **method_name** – The name of the method to call
- **wait_for** – Waits for all shards to sync before returning response
- **kwargs** – The arguments to pass to the method

Return type
dict

Returns
The result dictionary from ElasticSearch

static create_query_from_dict(*fields*)

Creates a query dict from a dictionary of fields and values

Parameters
fields (Dict[str, Union[str, int, float]]) – A dictionary of fields and values to search by

Return type
ESQuery

Returns
A query dict

async delete(*, *wait_for=False*, *routing=None*)

Deletes document from elasticsearch.

Parameters

- **wait_for** – Waits for all shards to sync before returning response - useful when writing tests. Defaults to False.
- **routing** (Optional[str]) – Shard routing value

Raises

- *esorm.error.NotFoundError* – Returned if document not found
- **ValueError** – Returned when id attribute missing from instance

classmethod from_es(*data*)

Returns an ESModel from an elasticsearch document that has _id, _source

Parameters
data (Dict[str, Any]) – Elasticsearch document that has _id, _source

Raises
esorm.error.InvalidResponseError – Returned when _id or _source is missing from data

Return type
Optional[TypeVar(TModel, bound= ESModel)]

Returns
The ESModel instance

async classmethod get(*id*, *, *routing=None*)

Fetches document and returns ESModel instance populated with properties.

Parameters

- **id** (Union[str, int, float]) – Document id
- **routing** (Optional[str]) – Shard routing value

Raises
esorm.error.NotFoundError – Returned if document not found

Return type
TypeVar(TModel, bound= ESModel)

Returns
ESModel object

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'forbid', 'populate_by_name': True, 'ser_json_bytes': 'base64', 'str_strip_whitespace': True, 'validate_assignment': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

model_post_init(*__context*)

This function is meant to behave like a BaseModel method to initialise private attributes.

It takes context as an argument since that's what pydantic-core passes when calling it.

Return type
None

Args:
self: The BaseModel instance. __context: The context.

async save(*, *wait_for=False*, *pipeline=None*, *routing=None*)

Save document into elasticsearch.

If document already exists, existing document will be updated as per native elasticsearch index operation. If model has id (Config.id_field or __id__), this will be used as the elasticsearch _id. The id field will be removed from the document before indexing. If no id is provided, then document will be indexed and elasticsearch will generate a suitable id that will be populated on the returned model.

Parameters

- **wait_for** – Waits for all shards to sync before returning response - useful when writing tests. Defaults to False.
- **pipeline** (Optional[str]) – Pipeline to use for indexing

- **routing** (Optional[str]) – Shard routing value

Return type
str

Returns
The new document's ID

async classmethod search(*query*, *, *page_size=None*, *page=None*, *sort=None*, *routing=None*, *res_dict=False*, ***kwargs*)

Search Model with query dict

Parameters

- **query** (*ESQuery*) – ElasticSearch query dict
- **page_size** (Optional[int]) – Pagination page size
- **page** (Optional[int]) – Pagination page num, 1st page is 1
- **sort** (Union[list, str, None]) – Name of field to be sorted, or sort term list of dict, if not specified, model's default sort will be used, or no sorting
- **routing** (Optional[str]) – Shard routing value
- **res_dict** (bool) – If the result should be a dict with id as key and model as value instead of a list of models
- **kwargs** – Other search API params

Return type
Union[List[TypeVar(TModel, bound= ESModel)], Dict[str, TypeVar(TModel, bound= ESModel)]]

Returns
The result list

async classmethod search_by_fields(*fields*, *, *page_size=None*, *page=None*, *sort=None*, *routing=None*, *aggs=None*, *res_dict=False*, ***kwargs*)

Search Model by fields as key-value pairs

Parameters

- **fields** (Dict[str, Union[str, int, float]]) – A dictionary of fields and values to search by
- **page_size** (Optional[int]) – Pagination page size
- **page** (Optional[int]) – Pagination page num, 1st page is 1
- **sort** (Union[list, str, None]) – Name of field to be sorted, or sort term list of dict, if not specified, model's default sort will be used, or no sorting
- **routing** (Optional[str]) – Shard routing value
- **aggs** (Optional[Dict[str, *ESAgg*]]) – Aggregations
- **res_dict** (bool) – If the result should be a dict with id as key and model as value instead of a list of models
- **kwargs** – Other search API params

Return type
List[TypeVar(TModel, bound= ESModel)]

Returns
The result list

async classmethod search_one(*query*, *, *routing=None*, ***kwargs*)

Search Model and return the first result

Parameters

- **query** (*ESQuery*) – ElasticSearch query dict
- **routing** (Optional[str]) – Shard routing value
- **kwargs** – Other search API params

Return type
Optional[TypeVar(TModel, bound= ESModel)]

Returns
The first result or None if no result

async classmethod search_one_by_fields(*fields*, *, *routing=None*, *aggs=None*, ***kwargs*)

Search Model by fields as key-value pairs and return the first result

Parameters

- **fields** (Dict[str, Union[str, int, float]]) – A dictionary of fields and values to search by
- **routing** (Optional[str]) – Shard routing value
- **aggs** (Optional[Dict[str, *ESAgg*]]) – Aggregations
- **kwargs** – Other search API params

Return type
Optional[TypeVar(TModel, bound= ESModel)]

Returns
The first result or None if no result

to_es(***kwargs*)

Generates a dictionary equivalent to what ElasticSearch returns in the '_source' property of a response.

It automatically removes the id field from the document if it is set in ESConfig.id_field to prevent duplication of the id field.

Parameters
kwargs – Pydantic's model_dump parameters

Return type
dict

Returns
The dictionary for ElasticSearch

class esorm.**ESModelTimestamp**(***data*)

Bases: *ESModel*

Model which stores *created_at* and *modified_at* fields automatcally.

created_at: Optional[datetime]

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'forbid', 'populate_by_name': True, 'ser_json_bytes': 'base64', 'str_strip_whitespace': True, 'validate_assignment': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'created_at': FieldInfo(annotation=Union[datetime, NoneType], required=False, default=None, description='Creation date and time'), 'modified_at': FieldInfo(annotation=Union[datetime, NoneType], required=False, default_factory=utcnow, description='Modification date and time')}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

Return type
None

modified_at: Optional[datetime]

async save(**, force_new=False, wait_for=False, pipeline=None, routing=None*)

Save document into elasticsearch.

If document already exists, existing document will be updated as per native elasticsearch index operation. If model has id (Meta.id_field or __id__), this will be used as the elasticsearch _id. The id field will be removed from the document before indexing. If no id is provided, then document will be indexed and elasticsearch will generate a suitable id that will be populated on the returned model.

Parameters

- **force_new** – Force creation of new document, it is assumed that document does not exist in elasticsearch
- **wait_for** – Waits for all shards to sync before returning response - useful when writing tests. Defaults to False.
- **pipeline** (Optional[str]) – Pipeline to use for indexing
- **routing** (Optional[str]) – Shard routing value

Return type
str

Returns
The new document's ID

esorm.**Field**(*default*, ***, *index=True*, *alias=None*, *title=None*, *description=None*, *exclude=None*, *include=None*, *frozen=False*, ***extra*)

Basic Field Info

Parameters

- **default** (Any) – since this is replacing the field's default, its first argument is used to set the default, use ellipsis (...) to indicate the field is required
- **index** (bool) – if this field should be indexed or not
- **alias** (Optional[str]) – the public name of the field

- **title** (Optional[str]) – can be any string, used in the schema
- **description** (Optional[str]) – can be any string, used in the schema
- **exclude** (Optional[bool]) – exclude this field while dumping. Takes same values as the include and exclude arguments on the .dict method.
- **include** (Optional[bool]) – include this field while dumping. Takes same values as the include and exclude arguments on the .dict method.
- **frozen** (bool) – if this field should be frozen or not
- **extra** – any additional keyword arguments will be added as is to the schema

Return type
FieldInfo

Returns
A field info object

exception esorm.**InvalidModelError**

Bases: Exception

Raised when a model is invalid.

exception esorm.**InvalidResponseError**

Bases: Exception

Raised when the response from Elasticsearch is invalid.

exception esorm.**NotFoundError**

Bases: Exception

Raised when a model is not found.

class esorm.**Pagination**(***data*)

Bases: BaseModel

Pagination parameters

callback: Optional[Callable[[int], Awaitable[None]]]

Callback after the search is done with the total number of hits

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'callback': FieldInfo(annotation=Union[Callable[[int], Awaitable[NoneType]], NoneType], required=False, default=None, description='Callback after the search is done with the total number of hits'), 'page': FieldInfo(annotation=int, required=False, default=1, description='The page number'), 'page_size': FieldInfo(annotation=int, required=False, default=10, description='The page size')}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

page: int

The page number

page_size: int

The page size

class esorm.**Sort**(***data*)

Bases: BaseModel

Sort parameters

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'sort': FieldInfo(annotation=Union[List[Dict[str, esorm.model.SortOrder]], str, NoneType], required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

sort: Union[List[Dict[str, SortOrder]], str, None]

async esorm.**connect**(*hosts*, **args*, *wait=False*, ***kwargs*)

Connect to ElasticSearch

Parameters

- **hosts** (Union[str, List[Union[str, Mapping[str, Union[str, int]], NodeConfig]]]) – ElasticSearch hosts to connect, either a list a mapping, or a single string
- **args** – Other AsyncElasticsearch arguments
- **wait** – Wait for AsyncElasticsearch to be ready
- **kwargs** – Other AsyncElasticsearch keyword arguments

Return type
Optional[AsyncElasticsearch]

Returns
AsyncElasticsearch client instance

esorm.**lazy_property**(*func*)

Decorator for lazy properties

Lazy properties computed after search from ES

Parameters
func (Callable[[], Awaitable[Any]]) – The async function to decorate

Returns
The decorated function

async esorm.**setup_mappings**(**_*, *debug=False*)

Create mappings for indices or try to extend it if there are new fields

3.2 Submodules

3.2.1 esorm.aggs module

ElasticSearch aggregation type definitions for ESORM

class esorm.aggs.**ESAgg**(**args*, ***kwargs*)

Bases: dict

Holds all types of aggregations supported

avg: *ESAggFieldParams*

Average aggregation

histogram: *ESAggHistogramParams*

Histogram aggregation

max: *ESAggFieldParams*

Maximum aggregation

min: *ESAggFieldParams*

Minimum aggregation

sum: *ESAggFieldParams*

Sum aggregation

terms: *ESAggTermParams*

Terms aggregation

class esorm.aggs.**ESAggBucketResponse**(**args*, ***kwargs*)

Bases: dict

Represents a single bucket in a bucket aggregation.

doc_count: **int**

The number of documents in this bucket.

key: **str**

The key of the bucket.

class esorm.aggs.**ESAggExtendedBounds**(**args*, ***kwargs*)

Bases: dict

Represents the parameters for extended bounds in Elasticsearch.

max: **int**

The maximum value.

min: **int**

The minimum value.

class esorm.aggs.**ESAggFieldParams**(**args*, ***kwargs*)

Bases: dict

Represents field parameter in Elasticsearch.

field: **str**

The field to aggregate on.

class esorm.aggs.**ESAggHistogramBucketresponse**(**args*, ***kwargs*)

Bases: dict

Represents a bucket in a histogram aggregation.

doc_count: int

The number of documents in this bucket.

key: float

Numeric key corresponding to the bucket's range.

class esorm.aggs.**ESAggHistogramParams**(**args*, ***kwargs*)

Bases: dict

Represents the parameters for a histogram aggregation in Elasticsearch.

extended_bounds: *ESAggExtendedBounds*

The extended bounds of the histogram.

field: str

The field to aggregate on.

interval: int

The interval of the histogram.

min_doc_count: int

The minimum number of documents in a bucket.

class esorm.aggs.**ESAggHistogramResponse**(**args*, ***kwargs*)

Bases: dict

Represents the response for a histogram aggregation.

buckets: List[*ESAggHistogramBucketresponse*]

A list of buckets in the histogram aggregation.

class esorm.aggs.**ESAggTermParams**(**args*, ***kwargs*)

Bases: dict

Represents the parameters for a terms aggregation in Elasticsearch.

field: str

The field to aggregate on.

order: Dict[str, str]

The order of the buckets.

size: int

The number of buckets to return.

class esorm.aggs.**ESAggTermsResponse**(**args*, ***kwargs*)

Bases: dict

Represents the response for a terms aggregation.

buckets: List[*ESAggBucketResponse*]

A list of buckets in the terms aggregation.

class esorm.aggs.**ESAggValueResponse**(**args*, ***kwargs*)

Bases: dict

Represents the response for an average, min, or max aggregation.

value: float

The average, min, or max value.

esorm.aggs.**ESAggs**

ElasticSearch aggregations type definition

alias of Dict[str, *ESAgg*]

esorm.aggs.**ESAggsResponse**

ElasticSearch aggregations response type definition

alias of Dict[str, Union[*ESAggValueResponse*, *ESAggTermsResponse*, *ESAggHistogramResponse*]]

3.2.2 esorm.bulk module

Bulk operation for ElasticSearch

class esorm.bulk.**ESBulk**(*wait_for=False*, ***bulk_kwargs*)

Bases: object

Bulk operation for ElasticSearch

async delete(*model*)

Add the model to the bulk for deletion :type model: TypeVar(TModel, bound= ESModel) :param model: The model to add for deletion

async save(*model*)

Add the model to the bulk for saving :type model: TypeVar(TModel, bound= ESModel) :param model: The model to add for saving

3.2.3 esorm.error module

This module contains all the exceptions that can be raised by ESORM.

exception esorm.error.**IndexDoesNotFoundError**

Bases: Exception

Raised when an index does not exist.

exception esorm.error.**InvalidModelError**

Bases: Exception

Raised when a model is invalid.

exception esorm.error.**InvalidResponseError**

Bases: Exception

Raised when the response from Elasticsearch is invalid.

exception esorm.error.**NotFoundError**

Bases: Exception

Raised when a model is not found.

3.2.4 esorm.esorm module

ElasticSearch ORM main module

async esorm.esorm.**connect**(*hosts*, **args*, *wait=False*, ***kwargs*)

Connect to ElasticSearch

Parameters

- **hosts** (Union[str, List[Union[str, Mapping[str, Union[str, int]], NodeConfig]]]) – ElasticSearch hosts to connect, either a list a mapping, or a single string
- **args** – Other AsyncElasticsearch arguments
- **wait** – Wait for AsyncElasticsearch to be ready
- **kwargs** – Other AsyncElasticsearch keyword arguments

Return type
Optional[AsyncElasticsearch]

Returns
AsyncElasticsearch client instance

3.2.5 esorm.fastapi module

FastaAPI utilities for ESORM

esorm.fastapi.**make_dep_pagination**(*default_page=1*, *default_page_size=10*, *set_headers=True*)

Create a pagination dependency with default values

Parameters

- **default_page** (int) – Default page number, the first page is 1
- **default_page_size** (int) – Default page size, the default is 10
- **set_headers** (bool) – Set X-Total-Hits header after search

Return type
callable

Returns
Pagination dependency

esorm.fastapi.**make_dep_sort**(***kwargs*)

Create a sort dependency with sort definitions

Parameters
kwargs (Union[List[Dict[str, dict]], Dict[str, any]]) – Sort definitions

Return type
callable

Returns
Sort dependency

esorm.fastapi.**set_max_page_size**(*max_page_size*)

Set the maximum page size for queries

Parameters
max_page_size (int) – The maximum page size

Returns
None

3.2.6 esorm.fields module

class esorm.fields.**Binary**

Bases: str

Stores binary data as base64 encoded strings

property bytes: bytes

classmethod validate_binary(*v*, *_*)

Return type
str

class esorm.fields.**Byte**

Bases: int

Byte Field

class esorm.fields.**Double**(*x=0*, */*)

Bases: float

Double Field

esorm.fields.**Field**(*default*, ***, *index=True*, *alias=None*, *title=None*, *description=None*, *exclude=None*, *include=None*, *frozen=False*, ***extra*)

Basic Field Info

Parameters

- **default** (Any) – since this is replacing the field's default, its first argument is used to set the default, use ellipsis (...) to indicate the field is required
- **index** (bool) – if this field should be indexed or not
- **alias** (Optional[str]) – the public name of the field
- **title** (Optional[str]) – can be any string, used in the schema
- **description** (Optional[str]) – can be any string, used in the schema
- **exclude** (Optional[bool]) – exclude this field while dumping. Takes same values as the include and exclude arguments on the .dict method.
- **include** (Optional[bool]) – include this field while dumping. Takes same values as the include and exclude arguments on the .dict method.
- **frozen** (bool) – if this field should be frozen or not
- **extra** – any additional keyword arguments will be added as is to the schema

Return type
FieldInfo

Returns
A field info object

class esorm.fields.**Float**(*x=0*, */*)

Bases: float

Float Field

class esorm.fields.**HalfFloat**(*x=0*, */*)

Bases: float

Half Float Field

class esorm.fields.**Integer**

Bases: int

Integer Field

class esorm.fields.**Keyword**

Bases: str

Keyword Field

class esorm.fields.**LatLon**(***data*)

Bases: BaseModel

Geo Point Field - Latitude and Longitude

lat: float

Latitude Coordinate

lon: float

Longitude Coordinate

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'lat': FieldInfo(annotation=float, required=True, description='Latitude Coordinate'), 'lon': FieldInfo(annotation=float, required=True, description='Longitude Coordinate')}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class esorm.fields.**Long**

Bases: int

Long Field

esorm.fields.**NumericField**(*default*, ***, *index=None*, *alias=None*, *gt=None*, *ge=None*, *lt=None*, *le=None*, *multiple_of=None*, *allow_inf_nan=None*, *max_digits=None*, *decimal_places=None*, *title=None*, *description=None*, *exclude=None*, *include=None*, *frozen=False*, ***extra*)

Numeric Field Info

Parameters

- **default** (Union[int, float]) – since this is replacing the field's default, its first argument is used to set the default, use ellipsis (...) to indicate the field is required

- **index** (Optional[bool]) – if this field should be indexed or not
- **alias** (Optional[str]) – the public name of the field
- **gt** (Optional[float]) – only applies to numbers, requires the field to be "greater than". The schema will have an exclusiveMinimum validation keyword
- **ge** (Optional[float]) – only applies to numbers, requires the field to be "greater than or equal to". The schema will have a minimum validation keyword
- **lt** (Optional[float]) – only applies to numbers, requires the field to be "less than". The schema will have an exclusiveMaximum validation keyword
- **le** (Optional[float]) – only applies to numbers, requires the field to be "less than or equal to". The schema will have a maximum validation keyword
- **multiple_of** (Optional[float]) – only applies to numbers, requires the field to be "a multiple of". The schema will have a multipleOf validation keyword
- **allow_inf_nan** (Optional[bool]) – only applies to numbers, allows the field to be NaN or infinity (+inf or -inf), which is a valid Python float. Default True, set to False for compatibility with JSON.
- **max_digits** (Optional[int]) – only applies to Decimals, requires the field to have a maximum number of digits within the decimal. It does not include a zero before the decimal point or trailing decimal zeroes.
- **decimal_places** (Optional[int]) – only applies to Decimals, requires the field to have at most a number of decimal places allowed. It does not include trailing decimal zeroes.
- **title** (Optional[str]) – can be any string, used in the schema
- **description** (Optional[str]) – can be any string, used in the schema
- **exclude** (Optional[bool]) – exclude this field while dumping. Takes same values as the include and exclude arguments on the .dict method.
- **include** (Optional[bool]) – include this field while dumping. Takes same values as the include and exclude arguments on the .dict method.
- **frozen** (bool) – if this field should be frozen or not
- **extra** – any additional keyword arguments will be added as is to the schema

Return type
FieldInfo

Returns
A field info object

class esorm.fields.**Short**

Bases: int

Short Field

class esorm.fields.**Text**

Bases: str

Text Field

esorm.fields.**TextField**(*default*, *, *index=True*, *alias=None*, *min_length=None*, *max_length=None*, *regex=None*, *title=None*, *description=None*, *exclude=None*, *include=None*, *frozen=False*, ***extra*)

Text Field Info

Parameters

- **default** (str) – since this is replacing the field's default, its first argument is used to set the default, use ellipsis (...) to indicate the field is required
- **index** (bool) – if this field should be indexed or not
- **alias** (Optional[str]) – the public name of the field
- **min_length** (Optional[int]) – only applies to strings, requires the field to have a minimum length. The schema will have a minLength validation keyword
- **max_length** (Optional[int]) – only applies to strings, requires the field to have a maximum length. The schema will have a maxLength validation keyword
- **regex** (Optional[str]) – only applies to strings, requires the field match against a regular expression pattern string. The schema will have a pattern validation keyword
- **title** (Optional[str]) – can be any string, used in the schema
- **description** (Optional[str]) – can be any string, used in the schema
- **exclude** (Optional[bool]) – exclude this field while dumping. Takes same values as the include and exclude arguments on the .dict method.
- **include** (Optional[bool]) – include this field while dumping. Takes same values as the include and exclude arguments on the .dict method.
- **frozen** (bool) – if this field should be frozen or not
- **extra** – any additional keyword arguments will be added as is to the schema

Return type
FieldInfo

Returns
A field info object

esorm.fields.**binary**
Binary type

alias of Union[*Binary*, str]

esorm.fields.**boolean**
alias of bool

esorm.fields.**byte**
Byte type

alias of Union[*Byte*, int]

esorm.fields.**double**
64 bit float (double) type

alias of Union[*Double*, float]

esorm.fields.**float16**
16 bit float type

alias of Union[*HalfFloat*, float]

esorm.fields.**float32**
32 bit float type

alias of Union[*Float*, float]

esorm.fields.**geo_point**
: Geo Point type

esorm.fields.**int32**
: 32 bit integer type

: alias of Union[*Integer*, int]

esorm.fields.**keyword**
: Keyword type

: alias of Union[*Keyword*, str]

esorm.fields.**long**
: 64 bit integer (long) type

: alias of Union[*Long*, int]

esorm.fields.**short**
: Short type

: alias of Union[*Short*, int]

esorm.fields.**text**
: Text type

: alias of Union[*Text*, str]

3.2.7 esorm.logger module

3.2.8 esorm.model module

This module contains the ESModel classes and related functions

class esorm.model.**ESBaseModel**(***data*)
: Bases: BaseModel

: Base class for Elastic

: It is useful for nested models, if you don't need the model in ES mappings

: **class ESConfig**
: Bases: object

: ESBaseModel Config

: This is just for lazy properties, to make ESBasemodel compatible with them

: **lazy_property_max_recursion_depth: int = 1**
: Maximum recursion depth of lazy properties

: **async calc_lazy_properties**()
: (re)Calculate lazy properties

: **model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}**
: A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'forbid', 'populate_by_name': True, 'ser_json_bytes': 'base64', 'str_strip_whitespace': True, 'validate_assignment': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class esorm.model.**ESModel**(***data*)

Bases: *ESBaseModel*

ElasticSearch Base Model

class ESConfig

Bases: object

ESModel Config

default_sort: Optional[List[Dict[str, Dict[str, str]]]] = None

Default sort

id_field: Optional[str] = None

The name of the 'id' field

index: Optional[str] = None

The index name

lazy_property_max_recursion_depth: int = 1

Maximum recursion depth of lazy properties

settings: Optional[Dict[str, Any]] = None

Index settings

async classmethod aggregate(*aggs*, ***, *query=None*, *routing=None*, ***kwargs*)

Aggregate Model with aggregation dict Before aggregation the model can be filtered by query dict.

Parameters

- **aggs** (Dict[str, *ESAgg*]) – Aggregation dict
- **query** (Optional[*ESQuery*]) – ElasticSearch query dict
- **routing** (Optional[str]) – Shard routing value
- **kwargs** – Other search API params

Return type

Dict[str, Union[*ESAggValueResponse*, *ESAggTermsResponse*, *ESAggHistogramResponse*]]

Returns

The result list

async classmethod all(***kwargs*)

Get all documents

Parameters

kwargs – Other search API params

Return type
List[TypeVar(TModel, bound= ESModel)]

Returns
The result list

async classmethod call(*method_name*, *, *wait_for=None*, ***kwargs*)

Call an elasticsearch method

This is a low level ES method call, it is not recommended to use this directly.

Parameters

- **method_name** – The name of the method to call
- **wait_for** – Waits for all shards to sync before returning response
- **kwargs** – The arguments to pass to the method

Return type
dict

Returns
The result dictionary from ElasticSearch

static create_query_from_dict(*fields*)

Creates a query dict from a dictionary of fields and values

Parameters
fields (Dict[str, Union[str, int, float]]) – A dictionary of fields and values to search by

Return type
ESQuery

Returns
A query dict

async delete(*, *wait_for=False*, *routing=None*)

Deletes document from elasticsearch.

Parameters

- **wait_for** – Waits for all shards to sync before returning response - useful when writing tests. Defaults to False.
- **routing** (Optional[str]) – Shard routing value

Raises

- *esorm.error.NotFoundError* – Returned if document not found
- **ValueError** – Returned when id attribute missing from instance

classmethod from_es(*data*)

Returns an ESModel from an elasticsearch document that has _id, _source

Parameters
data (Dict[str, Any]) – Elasticsearch document that has _id, _source

Raises
esorm.error.InvalidResponseError – Returned when _id or _source is missing from data

Return type
Optional[TypeVar(TModel, bound= ESModel)]

Returns
The ESModel instance

async classmethod get(*id*, *, *routing=None*)

Fetches document and returns ESModel instance populated with properties.

Parameters

- **id** (Union[str, int, float]) – Document id
- **routing** (Optional[str]) – Shard routing value

Raises
esorm.error.NotFoundError – Returned if document not found

Return type
TypeVar(TModel, bound= ESModel)

Returns
ESModel object

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'forbid', 'populate_by_name': True, 'ser_json_bytes': 'base64', 'str_strip_whitespace': True, 'validate_assignment': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

model_post_init(*__context*)

This function is meant to behave like a BaseModel method to initialise private attributes.

It takes context as an argument since that's what pydantic-core passes when calling it.

Return type
None

Args:
self: The BaseModel instance. __context: The context.

async save(*, *wait_for=False*, *pipeline=None*, *routing=None*)

Save document into elasticsearch.

If document already exists, existing document will be updated as per native elasticsearch index operation. If model has id (Config.id_field or __id__), this will be used as the elasticsearch _id. The id field will be removed from the document before indexing. If no id is provided, then document will be indexed and elasticsearch will generate a suitable id that will be populated on the returned model.

Parameters

- **wait_for** – Waits for all shards to sync before returning response - useful when writing tests. Defaults to False.

- **pipeline** (Optional[str]) – Pipeline to use for indexing
- **routing** (Optional[str]) – Shard routing value

Return type
str

Returns
The new document's ID

async classmethod search(*query*, *, *page_size=None*, *page=None*, *sort=None*, *routing=None*, *res_dict=False*, ***kwargs*)

Search Model with query dict

Parameters

- **query** (*ESQuery*) – ElasticSearch query dict
- **page_size** (Optional[int]) – Pagination page size
- **page** (Optional[int]) – Pagination page num, 1st page is 1
- **sort** (Union[list, str, None]) – Name of field to be sorted, or sort term list of dict, if not specified, model's default sort will be used, or no sorting
- **routing** (Optional[str]) – Shard routing value
- **res_dict** (bool) – If the result should be a dict with id as key and model as value instead of a list of models
- **kwargs** – Other search API params

Return type
Union[List[TypeVar(TModel, bound= ESModel)], Dict[str, TypeVar(TModel, bound= ESModel)]]

Returns
The result list

async classmethod search_by_fields(*fields*, *, *page_size=None*, *page=None*, *sort=None*, *routing=None*, *aggs=None*, *res_dict=False*, ***kwargs*)

Search Model by fields as key-value pairs

Parameters

- **fields** (Dict[str, Union[str, int, float]]) – A dictionary of fields and values to search by
- **page_size** (Optional[int]) – Pagination page size
- **page** (Optional[int]) – Pagination page num, 1st page is 1
- **sort** (Union[list, str, None]) – Name of field to be sorted, or sort term list of dict, if not specified, model's default sort will be used, or no sorting
- **routing** (Optional[str]) – Shard routing value
- **aggs** (Optional[Dict[str, *ESAgg*]]) – Aggregations
- **res_dict** (bool) – If the result should be a dict with id as key and model as value instead of a list of models
- **kwargs** – Other search API params

Return type
List[TypeVar(TModel, bound= ESModel)]

Returns
The result list

async classmethod search_one(*query*, *, *routing=None*, ***kwargs*)

Search Model and return the first result

Parameters

- **query** (*ESQuery*) – ElasticSearch query dict
- **routing** (Optional[str]) – Shard routing value
- **kwargs** – Other search API params

Return type
Optional[TypeVar(TModel, bound= ESModel)]

Returns
The first result or None if no result

async classmethod search_one_by_fields(*fields*, *, *routing=None*, *aggs=None*, ***kwargs*)

Search Model by fields as key-value pairs and return the first result

Parameters

- **fields** (Dict[str, Union[str, int, float]]) – A dictionary of fields and values to search by
- **routing** (Optional[str]) – Shard routing value
- **aggs** (Optional[Dict[str, *ESAgg*]]) – Aggregations
- **kwargs** – Other search API params

Return type
Optional[TypeVar(TModel, bound= ESModel)]

Returns
The first result or None if no result

to_es(***kwargs*)

Generates a dictionary equivalent to what ElasticSearch returns in the '_source' property of a response.

It automatically removes the id field from the document if it is set in ESConfig.id_field to prevent duplication of the id field.

Parameters
kwargs – Pydantic's model_dump parameters

Return type
dict

Returns
The dictionary for ElasticSearch

class esorm.model.**ESModelTimestamp**(***data*)

Bases: *ESModel*

Model which stores *created_at* and *modified_at* fields automatcally.

created_at: Optional[datetime]

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'forbid', 'populate_by_name': True, 'ser_json_bytes': 'base64', 'str_strip_whitespace': True, 'validate_assignment': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'created_at': FieldInfo(annotation=Union[datetime, NoneType], required=False, default=None, description='Creation date and time'), 'modified_at': FieldInfo(annotation=Union[datetime, NoneType], required=False, default_factory=utcnow, description='Modification date and time')}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

Return type
None

modified_at: Optional[datetime]

async save(*, *force_new=False*, *wait_for=False*, *pipeline=None*, *routing=None*)

Save document into elasticsearch.

If document already exists, existing document will be updated as per native elasticsearch index operation. If model has id (Meta.id_field or __id__), this will be used as the elasticsearch _id. The id field will be removed from the document before indexing. If no id is provided, then document will be indexed and elasticsearch will generate a suitable id that will be populated on the returned model.

Parameters

- **force_new** – Force creation of new document, it is assumed that document does not exist in elasticsearch
- **wait_for** – Waits for all shards to sync before returning response - useful when writing tests. Defaults to False.
- **pipeline** (Optional[str]) – Pipeline to use for indexing
- **routing** (Optional[str]) – Shard routing value

Return type
str

Returns
The new document's ID

class esorm.model.**Pagination**(**data*)

Bases: BaseModel

Pagination parameters

callback: Optional[Callable[[int], Awaitable[None]]]

Callback after the search is done with the total number of hits

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'callback': FieldInfo(annotation=Union[Callable[[int], Awaitable[NoneType]], NoneType], required=False, default=None, description='Callback after the search is done with the total number of hits'), 'page': FieldInfo(annotation=int, required=False, default=1, description='The page number'), 'page_size': FieldInfo(annotation=int, required=False, default=10, description='The page size')}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

page: int

The page number

page_size: int

The page size

class esorm.model.**Sort**(***data*)

Bases: BaseModel

Sort parameters

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'sort': FieldInfo(annotation=Union[List[Dict[str, esorm.model.SortOrder]], str, NoneType], required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

sort: Union[List[Dict[str, SortOrder]], str, None]

async esorm.model.**create_index_template**(*name*, ***, *prefix_name*, *shards=1*, *replicas=0*, ***other_settings*)

Create index template

Parameters

- **name** (str) – The name of the template
- **prefix_name** (str) – The prefix of index pattern
- **shards** – Number of shards
- **replicas** – Nuber of replicas
- **other_settings** (Any) – Other settings

Return type

object

Returns
The result object from ES

esorm.model.**lazy_property**(*func*)

Decorator for lazy properties

Lazy properties computed after search from ES

Parameters
func (Callable[[], Awaitable[Any]]) – The async function to decorate

Returns
The decorated function

esorm.model.**set_default_index_prefix**(*default_index_prefix*)

Set default index prefix we use for model and index creation

Parameters
default_index_prefix (str) – The default index prefix

esorm.model.**set_max_lazy_property_concurrency**(*concurrency*)

Set the maximum concurrency of processing lazy properties

If this is not set, the default is 16.

Parameters
concurrency (int) – The maximum concurrency

async esorm.model.**setup_mappings**(**_*, *debug=False*)

Create mappings for indices or try to extend it if there are new fields

3.2.9 esorm.query module

Elasticsearch query type definitions for ESORM

class esorm.query.**ESBool**(**args*, ***kwargs*)

Bases: dict

Bool query structure

boost: float

Boosting value for the query

filter: List[Union[*FieldRange*, *FieldTerm*, *FieldTerms*, *FieldMatch*, *FieldMatchPhrase*, *FieldExists*, *FieldWildcard*, *FieldPrefix*, *FieldFuzzy*, *FieldGeoDistance*, *FieldMatchAll*, *FieldESMatchNone*, *FieldBool*]]

Filter queries

minimum_should_match: Union[int, str]

Minimum number of should queries to match

must: List[Union[*FieldRange*, *FieldTerm*, *FieldTerms*, *FieldMatch*, *FieldMatchPhrase*, *FieldExists*, *FieldWildcard*, *FieldPrefix*, *FieldFuzzy*, *FieldGeoDistance*, *FieldMatchAll*, *FieldESMatchNone*, *FieldBool*]]

Must queries

must_not: List[Union[*FieldRange*, *FieldTerm*, *FieldTerms*, *FieldMatch*, *FieldMatchPhrase*, *FieldExists*, *FieldWildcard*, *FieldPrefix*, *FieldFuzzy*, *FieldGeoDistance*, *FieldMatchAll*, *FieldESMatchNone*, *FieldBool*]]

Must not queries

should: List[Union[*FieldRange*, *FieldTerm*, *FieldTerms*, *FieldMatch*, *FieldMatchPhrase*, *FieldExists*, *FieldWildcard*, *FieldPrefix*, *FieldFuzzy*, *FieldGeoDistance*, *FieldMatchAll*, *FieldESMatchNone*, *FieldBool*]]

Should queries

class esorm.query.**ESExists**(**args*, ***kwargs*)

Bases: dict

Represents an exists query to check if a field exists.

field: str

The field to check.

esorm.query.**ESFilter**

Represents filter queries in Elasticsearch

alias of List[Union[*FieldRange*, *FieldTerm*, *FieldTerms*, *FieldMatch*, *FieldMatchPhrase*, *FieldExists*, *FieldWildcard*, *FieldPrefix*, *FieldFuzzy*, *FieldGeoDistance*, *FieldMatchAll*, *FieldESMatchNone*, *FieldBool*]]

class esorm.query.**ESFuzzy**(**args*, ***kwargs*)

Bases: dict

Represents a fuzzy query for approximate matching in Elasticsearch.

boost: float

Optional boosting value for the query

fuzziness: Union[int, str]

Fuzziness value for the query

max_expansions: int

Maximum number of expansions for the query

prefix_length: int

Prefix length for the query

transpositions: bool

Whether to allow transpositions for the query

value: str

The value to search for.

class esorm.query.**ESGeoDistance**(**args*, ***kwargs*)

Bases: dict

Represents a geo_distance query for distance-based geospatial queries in Elasticsearch.

distance: Union[str, float]

The distance to search for.

distance_type: str

The distance type to use for the query.

location: Union[Dict[str, float], str]
The location to search from.

location_field: str
The field containing the location to search from.

validation_method: str
The validation method to use for the query.

class esorm.query.**ESMatch**(**args*, ***kwargs*)
Bases: dict

Represents the parameters for a match query in Elasticsearch.

analyzer: str
Optional analyzer to use for the query.

boost: Union[int, float]
Optional boosting value for the query.

fuzziness: Union[int, str]
Optional fuzziness value for the query.

max_expansions: int
Optional maximum number of expansions for the query.

operator: str
The operator to use for the query.

prefix_length: int
Optional prefix length for the query.

query: Union[str, int, float]
The value to search for.

zero_terms_query: str
Optional zero terms query for the query.

class esorm.query.**ESMatchAll**(**args*, ***kwargs*)
Bases: dict

Represents a match_all query for matching all documents in Elasticsearch.

boost: float
Optional boosting value for the query

class esorm.query.**ESMatchNone**(**args*, ***kwargs*)
Bases: dict

Represents a match_none query for matching no documents in Elasticsearch.

class esorm.query.**ESMatchPhrase**(**args*, ***kwargs*)
Bases: dict

Represents the parameters for a match_phrase query in Elasticsearch.

analyzer: str
Optional analyzer to use for the query.

boost: Union[int, float]

Optional boosting value for the query.

query: str

The value to search for.

slop: int

Optional slop value for the query.

esorm.query.**ESMust**

Represents must queries in Elasticsearch

alias of List[Union[*FieldRange*, *FieldTerm*, *FieldTerms*, *FieldMatch*, *FieldMatchPhrase*, *FieldExists*, *FieldWildcard*, *FieldPrefix*, *FieldFuzzy*, *FieldGeoDistance*, *FieldMatchAll*, *FieldESMatchNone*, *FieldBool*]]

esorm.query.**ESMustNot**

Represents must_not queries in Elasticsearch

alias of List[Union[*FieldRange*, *FieldTerm*, *FieldTerms*, *FieldMatch*, *FieldMatchPhrase*, *FieldExists*, *FieldWildcard*, *FieldPrefix*, *FieldFuzzy*, *FieldGeoDistance*, *FieldMatchAll*, *FieldESMatchNone*, *FieldBool*]]

class esorm.query.**ESPrefix**(**args*, ***kwargs*)

Bases: dict

Represents a prefix query for prefix matching in Elasticsearch.

boost: float

Optional boosting value for the query

rewrite: str

Optional, method used to rewrite the query (e.g., “constant_score”, “scoring_boolean”)

value: str

The prefix to search for.

class esorm.query.**ESQuery**(**args*, ***kwargs*)

Bases: dict

Elasticsearch query structure

aggs: Dict[str, *ESAgg*]

Aggregations query structure

bool: *ESBool*

Bool query structure

exists: *ESExists*

Exists query structure

fuzzy: Dict[str, *ESFuzzy*]

Fuzzy query structure

geo_distance: Dict[str, *ESGeoDistance*]

Geo distance query structure

match: Dict[str, *ESMatch*]

Match query structure

match_all: *ESMatchAll*
Match all query structure

match_none: *ESMatchNone*
Match none query structure

match_phrase: Dict[str, *ESMatchPhrase*]
Match phrase query structure

prefix: Dict[str, *ESPrefix*]
Prefix query structure

term: Dict[str, *ESTerm*]
Term query structure

wildcard: Dict[str, *ESWildcard*]
Wildcard query structure

class esorm.query.**ESRange**(**args*, ***kwargs*)
Bases: dict

Range query structure

gt: Union[int, float, str]
Greater than

gte: Union[int, float, str]
Greater than or equal

lt: Union[int, float, str]
Less than

lte: Union[int, float, str]
Less than or equal

esorm.query.**ESShould**
Represents should queries in Elasticsearch

alias of List[Union[*FieldRange*, *FieldTerm*, *FieldTerms*, *FieldMatch*, *FieldMatchPhrase*, *FieldExists*, *FieldWildcard*, *FieldPrefix*, *FieldFuzzy*, *FieldGeoDistance*, *FieldMatchAll*, *FieldESMatchNone*, *FieldBool*]]

class esorm.query.**ESTerm**(**args*, ***kwargs*)
Bases: dict

Represents the parameters for a term query in Elasticsearch.

boost: Union[int, float]
Optional boosting value for the query.

value: Union[str, int, float]
The value to search for.

class esorm.query.**ESWildcard**(**args*, ***kwargs*)
Bases: dict

Represents a wildcard query for pattern matching in Elasticsearch.

boost: float
Optional boosting value for the query

case_insensitive: bool

Optional, whether the query is case insensitive.

rewrite: str

Optional, method used to rewrite the query (e.g., "constant_score", "scoring_boolean")

value: str

The pattern to search for. e.g., "te?t" or "test*"

class esorm.query.**FieldBool**(**args*, ***kwargs*)

Bases: dict

Represents a bool query for combining other queries in Elasticsearch.

bool: *ESBool*

Bool query structure

class esorm.query.**FieldESMatchNone**(**args*, ***kwargs*)

Bases: dict

Represents a match_none query for matching no documents in Elasticsearch.

match_none: *ESMatchNone*

Match none query structure

class esorm.query.**FieldExists**(**args*, ***kwargs*)

Bases: dict

Represents an exists query to check if a field exists in Elasticsearch.

exists: *ESExists*

Exists query structure

class esorm.query.**FieldFuzzy**(**args*, ***kwargs*)

Bases: dict

Represents a fuzzy query for approximate matching in Elasticsearch.

fuzzy: Dict[str, *ESFuzzy*]

Fuzzy query structure

class esorm.query.**FieldGeoDistance**(**args*, ***kwargs*)

Bases: dict

Represents a geo_distance query for distance-based geospatial queries in Elasticsearch.

geo_distance: Dict[str, *ESGeoDistance*]

Geo distance query structure

class esorm.query.**FieldMatch**(**args*, ***kwargs*)

Bases: dict

Represents a match query for matching based on the provided text in Elasticsearch.

match: Dict[str, *ESMatch*]

Match query structure

class esorm.query.**FieldMatchAll**(**args*, ***kwargs*)

Bases: dict

Represents a match_all query for matching all documents in Elasticsearch.

match_all: *ESMatchAll*

Match all query structure

class esorm.query.**FieldMatchPhrase**(**args*, ***kwargs*)

Bases: dict

Represents a match_phrase query for exact phrase matching in Elasticsearch.

match_phrase: **Dict[str, *ESMatchPhrase*]**

Match phrase query structure

class esorm.query.**FieldPrefix**(**args*, ***kwargs*)

Bases: dict

Represents a prefix query for prefix matching in Elasticsearch.

prefix: **Dict[str, *ESPrefix*]**

Prefix query structure

class esorm.query.**FieldRange**(**args*, ***kwargs*)

Bases: dict

Range qyery field

range: **Dict[str, *ESRange*]**

Range query structure

class esorm.query.**FieldTerm**(**args*, ***kwargs*)

Bases: dict

Represents a term query for exact value matching in Elasticsearch.

term: **Dict[str, *ESTerm*]**

Term query structure

class esorm.query.**FieldTerms**(**args*, ***kwargs*)

Bases: dict

Represents a terms query for exact value matching in Elasticsearch.

terms: **Dict[str, List[Union[str, int, float]]]**

Terms query structure

class esorm.query.**FieldWildcard**(**args*, ***kwargs*)

Bases: dict

Represents a wildcard query for pattern matching in Elasticsearch.

wildcard: **Dict[str, *ESWildcard*]**

Wildcard query structure

3.2.10 esorm.response module

This module contains type definitions for the response from Elasticsearch.

class esorm.response.**ESResponse**(**args*, ***kwargs*)

Bases: dict

Represents the overall structure of an Elasticsearch response.

aggregations: Dict[str, Union[*ESAggValueResponse*, *ESAggTermsResponse*, *ESAggHistogramResponse*]]

The aggregations section of the response.

hits: *Hits*

The hits section of the response.

timed_out: bool

Whether the query timed out.

took: int

The time in milliseconds it took to execute the query.

class esorm.response.**Hit**(**args*, ***kwargs*)

Bases: dict

Represents a single hit (result) from Elasticsearch.

class esorm.response.**Hits**(**args*, ***kwargs*)

Bases: dict

Represents the hits section of the Elasticsearch response.

hits: List[*Hit*]

List of hits.

max_score: Optional[float]

The maximum score of the hits.

total: Dict[str, int]

The total number of hits.

3.2.11 esorm.utils module

Utility functions

esorm.utils.**camel_case**(*snake_str*, *capitalize_first=False*)

Convert to camel case

Parameters

- **snake_str** (str) – The string to convert to camel case
- **capitalize_first** (bool) – Capitalize the first letter

Returns

Converted string

esorm.utils.**snake_case**(*camel_str*)

Convert to snake case

Parameters
camel_str (str) – The string to convert to snake case

Returns
Converted string

esorm.utils.**utcnow**()

Get current UTC time

Returns
Current UTC time

3.2.12 esorm.watcher module

ElasticSearch Watcher support for ESORM

class esorm.watcher.**Action**(**args*, ***kwargs*)

Bases: dict

Action definition

email: Dict[str, Any]

index: Dict[str, Any]

logging: Dict[str, Any]

pagerduty: Dict[str, Any]

slack: Dict[str, Any]

throttle_period: str

transform: Dict[str, *Transform*]

webhook: *ActionWebhook*

class esorm.watcher.**ActionWebhook**(**args*, ***kwargs*)

Bases: dict

Action webhook definition

auth: Dict[str, Any]

body: str

connection_timeout: str

headers: Dict[str, Any]

host: str

method: str

params: Dict[str, Any]

path: str

port: int

proxy: Dict[str, Any]

read_timeout: str

retries: int

retry_on_status: List[int]

scheme: str

ssl: Dict[str, Any]

timeout: str

webhook: Dict[str, Any]

class esorm.watcher.**ArrayCompare**(**args*, ***kwargs*)

Bases: dict

Array compare definition

eq: Any

gt: Union[int, float, str, Dict[str, Any]]

gte: Union[int, float, str, Dict[str, Any]]

lt: Union[int, float, str, Dict[str, Any]]

lte: Union[int, float, str, Dict[str, Any]]

not_eq: Any

path: str

class esorm.watcher.**Body**(**args*, ***kwargs*)

Bases: dict

Body definition

query: *ESQuery*

size: int

sort: Dict[str, *Order*]

class esorm.watcher.**Compare**(**args*, ***kwargs*)

Bases: dict

Compare definition

eq: Any

gt: Union[int, float, str, Dict[str, Any]]

gte: Union[int, float, str, Dict[str, Any]]

lt: Union[int, float, str, Dict[str, Any]]

lte: Union[int, float, str, Dict[str, Any]]

not_eq: Any

class esorm.watcher.**Condition**(**args*, ***kwargs*)

Bases: dict

Condition definition

always: *EmptyDict*

array_compare: Dict[str, Any]

compare: Dict[str, *Compare*]

never: *EmptyDict*

script: Dict[str, Any]

class esorm.watcher.**DeleteWatcher**

Bases: *Watcher*

Watcher for deleting documents matching a query

class esorm.watcher.**EmptyDict**(**args*, ***kwargs*)

Bases: dict

Empty dict definition

class esorm.watcher.**Order**(**args*, ***kwargs*)

Bases: dict

Order definition

order: str

class esorm.watcher.**Request**(**args*, ***kwargs*)

Bases: dict

Request definition

body: *ESQuery*

indices: Union[List[str], str]

tepmlate: Dict[str, Any]

class esorm.watcher.**Schedule**(**args*, ***kwargs*)

Bases: dict

Schedule definition

interval: str

class esorm.watcher.**Search**(**args*, ***kwargs*)

Bases: dict

Search definition

extract: List[str]

request: *Request*

class esorm.watcher.**Transform**(**args*, ***kwargs*)

Bases: dict

Transform definition

chain: List[Dict[str, Any]]

script: Dict[str, Any]

search: Dict[str, Any]

class esorm.watcher.**Trigger**(**args*, ***kwargs*)

Bases: dict

Trigger definition

schedule: *Schedule*

class esorm.watcher.**Watcher**

Bases: object

Watcher definition

actions: Optional[Dict[str, *Action*]] = None

condition: Optional[*Condition*] = None

input: Optional[*Search*] = None

metadata: Optional[Dict[str, Any]] = None

to_es()

trigger: Optional[*Trigger*] = None

class esorm.watcher.**WatcherMeta**(*name*, *bases*, *attrs*)

Bases: type

Watcher metaclass

async esorm.watcher.**setup_watchers**(*_*, *debug=False*)

Setup watchers :param _: Unused :type debug: :param debug: Whether to print the watcher definition

CHAPTER

FOUR

CHANGELOG

4.1 v0.3.2

Released on: 2024-05-03

- Lazy properties are now work with ESBaseModel too
- Lazy properties are now supports nested documents (truly recursive)
- Python 3.8 support back

4.2 v0.3.1

Released on: 2024-04-29

The most important change is to introduce ESBaseModel.

Full Changelog: https://github.com/wallneradam/esorm/compare/v0.3.0...v0.3.1

4.3 v0.3.0

Released on: 2024-01-25

- Replaced deprecated datetime.utcnow
- .all() method for getting all documents in an index
- Reworked lazy properties, which has recursion protection
- Improved documentation: - pagination and sorting - added advanced.md, which contains advanced features documentation - Fixed dark mode colors of reference
- Improved unit tests

4.4 v0.2.1

Released on: 2024-01-22

Aggregation TypeDicts are refactored.

4.5 v0.2.0

Released on: 2024-01-18

- Aggregation support
- Improved documentation

4.6 v0.1.2

Released on: 2024-01-17

Some minor fixes

4.7 v0.1.1

Released on: 2023-11-01

1st release

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

INDEX

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z